Exploring 3D Gaussian Splatting

Course Name: Machine Vision

Course Number: 16:332:561

Year: Fall 2025

Machine Vision Term Project: Project Report

Professor: Yuqian Zhang

Student Name and RUID: Melis E. Durgut - 212000878, Aarti Rao - 207009762

Date Submitted: 14 December 2025

Project Link:

https://github.com/turk1shdel1ght/Machine-Vision-Final-Project-3DGS-

Table of Contents

Section I: Objective

The objective of this project is to implement a 3D Gaussian Splatting pipeline to reconstruct realistic 3D scenes from multiple 2D images. The project aims to explore how Gaussian splats can be used as an efficient alternative to traditional neural radiance fields (NeRFs) for real-time, photorealistic scene rendering.

Section II: Overview / Introduction

Recent advances in computer vision and graphics have made it possible to reconstruct photorealistic 3D scenes from 2D image collections. While Neural Radiance Fields, or NeRFs have been a dominating presence in this field, their reliance on dense volumetric rendering and neural networks at every pixel leads to a high training time and much slower rendering. In order to address these limitations, we decided to experiment with a concept known as 3D Gaussian Splatting.

3D Gaussian Splatting is a recent development in computer vision and graphics which represents a 3D scene as a collection of anisotropic Gaussian primitives. Unlike NeRFs, Gaussian allows for explicit scene representation and fast rasterization, while enabling real time rendering and maintaining high visual fidelity. The primary aim of this project is to build an end to end pipeline that takes a set of RGB images of an object or environment along with their camera poses, and reconstructs a 3D scene using Gaussian splats that can be rendered from novel viewpoints. The pipeline would involve concepts like data preprocessing, Gaussian initialization, differentiable rendering, parameter optimization, and quantitative evaluation against ground truth images.

Section III: Related Work

This project relates to two major works in neural rendering: Neural Radiance Fields (NeRF) and 3D Gaussian Splatting. Together, both of these topics form the foundation for our 3D scene reconstruction project.

Section III(a): NeRF: Representing scenes as neural radiance fields for view synthesis [1]

In this paper, the authors propose a method known as Neural Radiance Fields, or NeRF, which optimizes an underlying continuous volumetric scene function using a sparse set of input views, ultimately enabling the synthesis of complex scenes. NeRF represents a scene as a continuous 5D coordinate system that maps 3D coordinates and the viewing direction to the volume density and radiance at the spatial location. It uses a method known as multilayer perceptron, or MLP, and volume rendering to render photorealistic novel views of scenes that consist of a complicated appearance. While NeRF yields effective results, it is computationally expensive, slow to train, and not able to complete real time rendering due to its architecture. However, it does provide a strong foundation for various research topics, including 3D Gaussian Splatting. Our project builds on this work by using the idea of novel view synthesis with RGB images.

Section III(b): 3D Gaussian Splatting for Real-Time Radiance Field Rendering [2]

In this paper, the authors propose a method known as 3D Gaussian Splatting, which is essentially a radiance field representation composed of anisotropic 3D Gaussian primitives. This method achieves real time rendering of radiance fields while maintaining competent optimization levels and novel view synthesis. The method proposed by the authors of this paper involves three key elements needed for 3D Gaussian Splatting.

The first element is scene representation using anisotropic 3D Gaussians. In this element, each point in the scene is converted to an anisotropic 3D Gaussian ellipsoid that preserves the properties needed to accurately approximate the local geometry and appearance. Second, the method performs interleaved optimization and density control of the 3D Gaussians, which optimizes the anisotropic covariance to achieve a more accurate representation of the scene. Thirdly, the authors develop a fast visibility-aware rendering algorithm that supports anisotropic splatting and accelerates training while enabling real time rendering. This element is then tested on several datasets to showcase its rendering and performance quality. This approach builds upon NeRF, and introduces various key advantages such as real time rendering and faster optimization. Our project directly builds on this work by using the explicit Gaussian representation and pipeline used by the authors of this paper.

Section IV: Datasets Used

Section IV(a): Voxel51 Gaussian Splatting Dataset[3]

The Voxel51 Gaussian Splatting Dataset[3] provides several 3D Gaussian Splatting scene reconstructions using the method introduced in the related works. Each scene has the reference image along with its Gaussian-splat representation at different optimization stages and iterations. In our project, we are using this dataset for initial evaluation on how 3D-Gaussian Splatting is supposed to work in order to get a rough idea of our pipeline.

Dataset 2: Lego Robot dataset

The Lego Robot dataset is created by us for this project specifically. The reason behind this dataset was the memory constraints of Google Colab, which is the platform we used for this project. The dataset consists of 50 images of the lego version of the robot Wall-E from different angles. The object is centered in front of a white background to make things easier for feature extraction.

Section V: Methodology

This is the methodology we are using in our project. This pipeline successfully helps us apply 3D Gaussian Splatting techniques to the dataset.

0. **Initial Research:** Explore 3D Gaussian on a different dataset to study results.
1. **Data Collection and Preparation:** Use multi-view images of static objects or scenes from publicly available datasets. Extract camera intrinsics and extrinsics using Structure-from-Motion (SfM) tools.
2. **3D Gaussian Splat Representation:** Represent each 3D point as an anisotropic Gaussian ellipsoid parameterized by position, covariance, color, and opacity. Initialize Gaussians from sparse point clouds generated by SfM.
3. **Rendering and Optimization:** Implement differentiable rasterization of Gaussians to render novel views. Optimize Gaussian parameters using gradient descent to minimize the photometric error between rendered and ground-truth images.
4. **Evaluation:** Compare rendered novel views to ground-truth images using metrics such as PSNR, SSIM, and LPIPS.

Section VI: Experimentative Results

The following section shows our experimentative results of our pipeline. We split the evaluation in two parts. The first part utilized 30 images in our dataset. However, we noticed that it was not enough to get quantitative results. Hence, we expanded our dataset to 50 images, making it almost twice as big. Once we got more definitive results, we moved to part 4 of our pipeline.

Section VI(a): Part 0 Results (Initial Research)

Unfortunately, there are not many publicly available 3D Gaussian Splatting datasets. We used the https://huggingface.co/datasets/Voxel51/gaussian_splatting dataset to try and understand better and experiment with 3D Gaussian Splatting and we are also using it to train our model. We started with experimenting with the datasets before the training process. The results of our experiments are shown below:

Figure 1: Result of the initial research

Figure 2: Result of the initial research (different angle)

Steps 2 and 3 of our results are split into two parts. Part 1 discusses our results for the 30 Image Dataset, while Part 2 discusses our results for the 50 Image Dataset.

Section VI(b): Part 1 Results → 30 Image Dataset

Initially, our dataset had images with busy backgrounds, which made things harder for COLMAP to pick up on the features that we wanted. So, we had to change the background to a more simple white one.

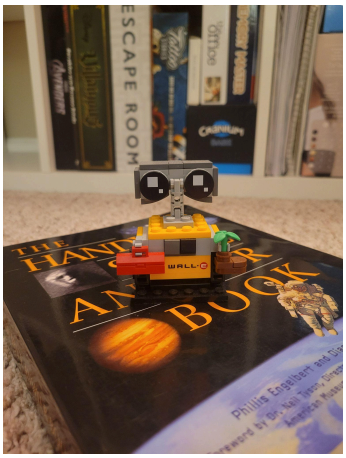

Figure 3: Example image from the previous dataset

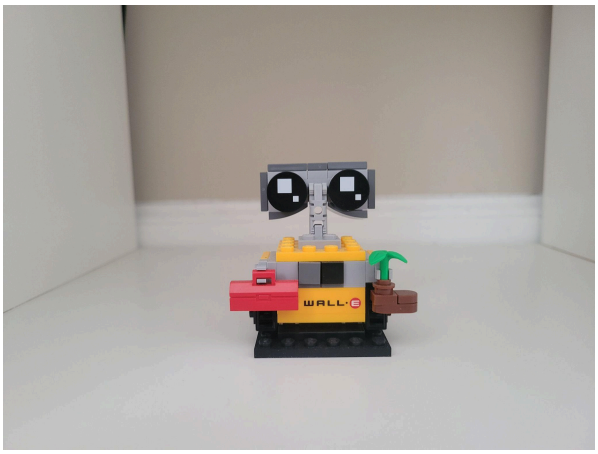

Figure 4: Example image from the current dataset

When we tested our code with the second dataset with the simpler background, we got better results. The point cloud looked more like what we expected and was way better than our initial results and the loss kept decreasing.

```
Cameras: 1

Images: 14

3D Points: 538
```

```
Iter 0, Loss 0.437249
Iter 20, Loss 0.435739
Iter 40, Loss 0.434755
Iter 60, Loss 0.434263
Iter 80, Loss 0.434073
Iter 100, Loss 0.433982
Iter 120, Loss 0.433915
Iter 140, Loss 0.433866
Iter 160, Loss 0.433826
Iter 180, Loss 0.433795
```

Figure 5: Loss results over iterations for 30 image dataset

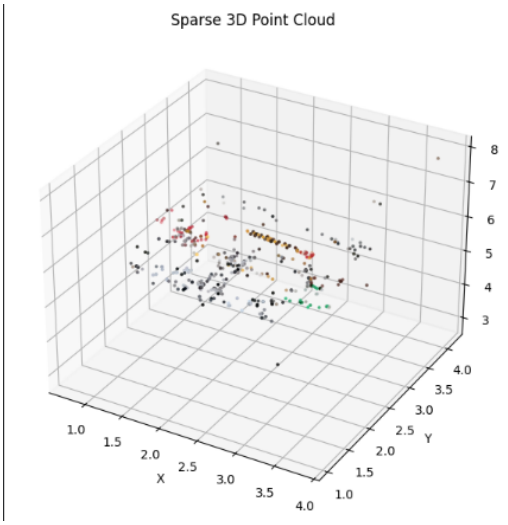

Figure 6: 3D Point Cloud for 30 image dataset

When we rendered, we got some initial results that were not centered but they looked somewhat like the ground truth images.

Figure 7: First rendered image of the 30 image dataset

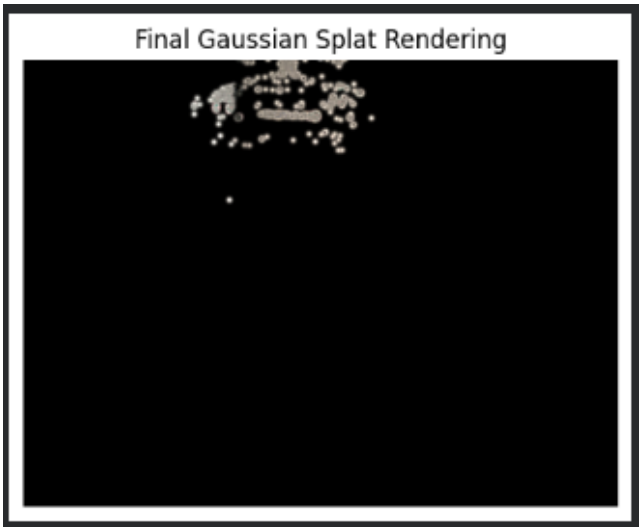

Figure 8: Final rendered image of the 30 image dataset

Part VI(c): Part 2 Results → 50 Image Dataset

The results we got with the 30 image dataset were good but they weren't great so we wanted to get more 3D points with more images. We expanded the dataset to 50 images and ran our code - which gave us a more satisfactory 3D point cloud and better loss.

```
Cameras: 1

Images: 36

3D Points: 2117
```

```
Iter 0, Loss 0.439078
Iter 20, Loss 0.435443
Iter 40, Loss 0.433517
Iter 60, Loss 0.432464
Iter 80, Loss 0.431943
Iter 100, Loss 0.431679
Iter 120, Loss 0.431506
Iter 140, Loss 0.431375
Iter 160, Loss 0.431269
Iter 180, Loss 0.431180
```

Figure 9: Loss results over iterations for 50 image dataset

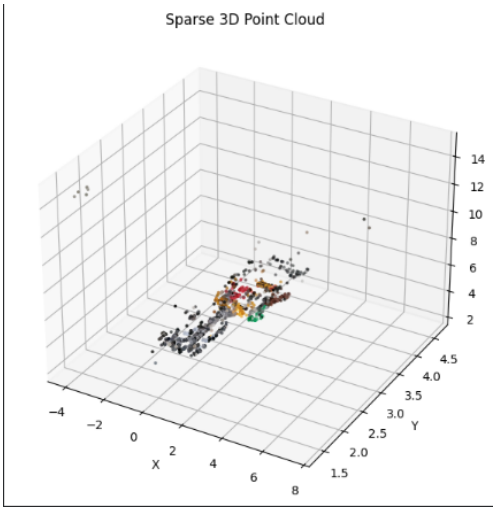

Figure 10: 3D Point Cloud for 50 image dataset

When we rendered the 50 image dataset, we finally got an image really close to the ground truth images. We still had a problem with the rendered image being on the top of the page, so we had to change some parameters to make it centered.

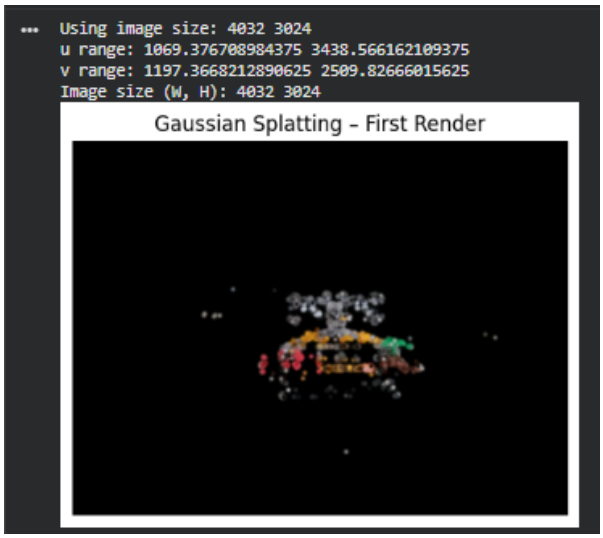

Figure 11: First rendered image of the 50 image dataset

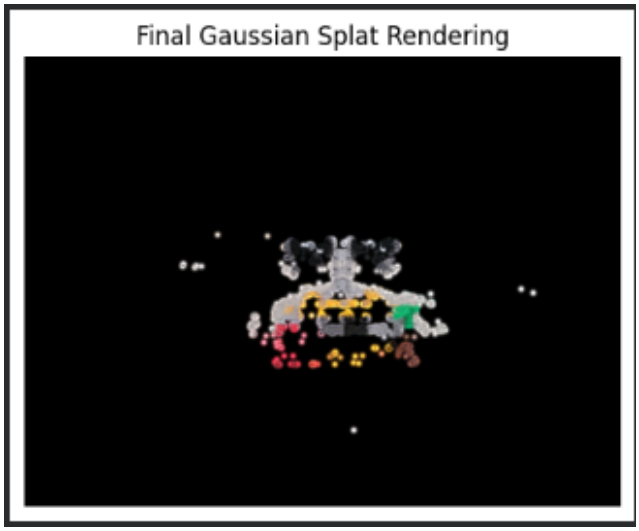

Figure 12: Final rendered image of the 50 image dataset

Part IV(d): Evaluation

To evaluate the accuracy of our project, we compared our rendered images to ground-truth images using metrics such as PSNR, SSIM, and LPIPS. Our Peak-to-Signal Ratio was 3.59 dB, Structural Similarity Index was 0.0188, and Learned Perceptual Image Patch Similarity was 0.8670. These values basically mean that the pixel difference between the rendered images and the ground truth images are big. However, when we look at the rendered images and the ground truth, we can see that they are very similar. Thus, we think we ran into a scaling/misalignment issue but we couldn't mess with any camera parameters as it affected the whole rendering process.

Section VII: GPUs Used

Section VII(a): NVIDIA T4

Initially, the first gpu we used was the NVIDIA T4 GPU, which is for free and accessible on Google Colab. This GPU contains 16 GB VRAM, which was not enough for our project. Hence, we ran into a lot of Out of Memory errors, and had to modify our code in ways that did not yield proper results.

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   30C    P8             9W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Figure 13: NVIDIA T4 Architecture

Section VII(b): NVIDIA A100

Once we realized that our GPU was not very efficient, we switched to the NVIDIA A100 GPU, which has 80 gb VRAM. This provided us with much more storage, and we were able to use our original code without getting any out of memory errors. The architecture of the A100 GPU is shown below:

```
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA A100-SXM4-80GB          Off |   00000000:00:05.0 Off |                    0 |
| N/A   32C    P0             52W /  400W |       0MiB /  81920MiB |      0%      Default |
|                                         |                        |             Disabled |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

Figure 14: NVIDIA A100 Architecture

Section VIII: Challenges:

1. **Hardware/Software challenges:** finding a GPU strong enough to run COLMAP and then rendering was extremely difficult, especially because rendering works best on Linux computers. Since we didn't have Linux computers and our virtual machine setups couldn't access our GPUs, we had to utilize Google Colab (which has its own issues such as runtime, GPU and memory limitations, etc)
2. **Dataset limitations:** We couldn't use already available datasets because those datasets were too big to upload to Google Colab and creating a dataset from scratch multiple times to perfect it for this project was time consuming.
3. **Feature Extraction:** We tried multiple methods for feature extraction but most of them were not Google Colab compatible or there was a version matching issue. COLMAP was the one that worked best and most accurately.
4. **Centering the Rendered Image:** Our rendered image was always on top of the little display and figuring out the issue was a challenge.

Section IX: Peer Review Evaluation

Our peer reviewers did a thorough job in giving us feedback to further improve our project. Here were some of the pointers they gave and how we utilized them to improve our report.

Section IX(a): Strong Points

1. **Clear Motivation:** The paper explicitly identifies the problem space (the trade-off between visual fidelity and rendering speed in NeRFs) and effectively motivates the choice of 3D Gaussian Splatting as a solution.

2. **Literature Context:** The review of related work is accurate and highlights the specific contributions of the foundational papers (Mildenhall et al. and Kerbl et al.) that this project builds upon.
3. **Defined Evaluation Metrics:** The authors have correctly identified the standard quantitative metrics for this domain (PSNR, SSIM, LPIPS), which will allow for objective assessment of their results once the pipeline is fully operational.
4. **Visual Evidence:** The inclusion of preliminary visualizations (the train dataset) demonstrates that the authors have successfully engaged with the data ingestion phase of the pipeline.

Section IX(b): Weaknesses

1. **Lack of Quantitative Results:** As a project report draft, the current submission is missing the core experimental data. The ”Experimental Results” section describes an intent to train but does not yet present tables, charts, or error metrics. This makes it impossible to evaluate the success of the implementation at this stage. **– Updated the results section with experimental data and evaluation metrics.**
2. **Future Tense/Proposal Style:** Much of the methodology is written in the future tense (for example, ”The pipeline would involve concepts...”). A final report should document what was done rather than what will be done. **– Changed the tone of the paper to fit the final report.**
3. **Vague Technical Specifics:** While the broad steps are defined, specific implementation details are missing. For example, the methodology mentions ”Structure-from-Motion (SfM) tools” but does not specify which library (COLMAP, etc.) is being used. **– Explained which method is used in each step clearly.**

Section IX(c): Additional Feedback

1. **Prioritize Numerical Results:** The most critical next step is to generate a table comparing your PSNR/SSIM scores against a baseline. Even if the results are not perfect, having the data is essential for a complete report. **– Updated the report with more data and results.**
2. **Image Captions:** Make sure all figures (like the train images in Section VI) have descriptive captions explaining exactly what the viewer is looking at (for example, ”Figure 1: Initial point cloud initialization” vs ”Figure 2: Rendered view at iteration 7,000”). **– Added descriptive captions to each image.**
3. **Tone Adjustment:** Shift the writing perspective from a ”proposal” tone to a ”report” tone. Instead of saying ”The project aims to do...”, say ”We implemented a pipeline that does...” **–Adjusted the tone to fit the final report.**

Section X: Conclusion and Next Steps

Our project primarily focuses on applying 3D Gaussian Splatting to reconstruct scenes from RGB images. After preparing the data and initializing the Gaussian representations, the next step of the project is to render the scenes using differentiable splatting and begin optimizing the Gaussians to improve the accuracy and visual quality of the reconstructed views. This stage will involve refining parameters such as position, covariance, color, and opacity to progressively achieve a more faithful representation of the underlying scene.

In terms of Next Steps, our goal would be to test this implementation against NeRF based methodologies, and to try and test on a much larger scale. We also want to try video rendering or interactive rendering with our dataset and larger datasets.

Section XI: Works Cited

[1] B. Kerbl, G. Kopanas, T. Leimkühler, and G. Drettakis, “3D Gaussian Splatting for Real-Time Radiance Field Rendering,” *ACM Trans. Graph.*, vol. 42, no. 4, 2023.

[2] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “NeRF: Representing scenes as neural radiance fields for view synthesis,” *Proc. ECCV*, 2020

[3] Voxel51, “Gaussian Splatting Dataset,” Hugging Face, 2023.

[Online]. Available: https://huggingface.co/datasets/Voxel51/gaussian_splatting